Tutorial for the SPARK Parallelizing High-Level Synthesis Framework Version 1.1

Sumit Gupta

Center for Embedded Computer Systems
University of California at San Diego and Irvine
sumitg@cecs.uci.edu
http://mesl.ucsd.edu/spark

Copyright © 2003-2004 The Regents of the University of California.
All Rights Reserved.

April 14, 2004

Contents

1 About this Tutorial **2**

1.1 Reporting Bugs . . . 2
1.2 Acknowledgments . . . 2
1.3 Change Log . . . 2
1.4 Copyright . . . 3
1.5 Disclaimer . . . 3

2 Downloading and Executing the *Spark* Tutorial **4**

2.1 Downloading and Setting up the Tutorial . . . 4
2.2 Synthesizing the Motion Compensation Algorithm . . . 5
2.3 Functional Verification of Output C Generated by *Spark* . . . 7
2.4 Viewing the Graphical Output of *Spark* . . . 8
2.5 Logic Synthesis of VHDL Output of *Spark* . . . 8
2.6 Logic Synthesis using non-Synopsys Logic Synthesis Tools . . . 9

Chapter 1

About this Tutorial

This tutorial demonstrates how to use the *Spark* parallelizing high-level synthesis software tool with an example. Through the tutorial, you will learn how to invoke *Spark*, how to sometimes modify C code to make it synthesizable by *Spark*.

1.1 Reporting Bugs

The *Spark* distribution comes with no official bug fixing support or maintenance and we are not obliged to provide any updates or modifications. However, you may report bugs to:

spark@ics.uci.edu

Subject: BUG: Brief description of bug

1.2 Acknowledgments

The *Spark* framework was developed by Sumit Gupta with major contributions to the underlying framework by Nick Savoiu. Mehrdad Reshadi and Sunwoo Kim also contributed to the code base. Professors Rajesh Gupta, Nikil Dutt and Alex Nicolau led the *Spark* project. This project was funded by Semiconductor Research Corporation and Intel Incorporated.

1.3 Change Log

Date	Changes
1/12/04	Introduced documentation for the newly introduced Windows version of SPARK. Also, some additional notes in the VHDL section since output VHDL is now synthesizable by Xilinx XST
09/1/03	First release of document

1.4 Copyright

The *Spark* software is Copyright ©2003-2004 The Regents of the University of California. All Rights Reserved.

Permission to use, copy, modify, and distribute this software and its documentation for educational, research and non-profit purposes, without fee, and without a written agreement is hereby granted, provided that the above copyright notice, this paragraph and the following three paragraphs appear in all copies.

Permission to incorporate this software into commercial products or for use in a commercial setting may be obtained by contacting:
Technology Transfer Office
9500 Gilman Drive, Mail Code 0910
University of California
La Jolla, CA 92093-0910
(858) 534-5815
invent@ucsd.edu

1.5 Disclaimer

- The *Spark* software program and the documentation is copyrighted by the Regents of the University of California. The following terms apply to all files associated with the software unless explicitly disclaimed in individual files.

- The software program and documentation are supplied ”as is”, without any accompanying services from The Regents. The Regents does not warrant that the operation of the program will be uninterrupted or error-free. The end-user understands that the program was developed for research purposes and is advised not to rely exclusively on the program for any reason.

- In no event shall the University of California be liable to any party for direct, indirect, special, incidental, or consequential damages, including lost profits, arising out of the use of this software and its documentation, even if the University of California has been advised of the possibility of such damage. The University of California specifically disclaims any warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. the software provided hereunder is on an ”as is” basis, and the University of California has no obligations to provide maintenance, support, updates, enhancements, or modifications.

Chapter 2

Downloading and Executing the *Spark* Tutorial

In this chapter, we explain how to download, setup and execute the tutorial. The sample design we have used for this tutorial is the *Berkeley MPEG Player* [1]. The MPEG player is representative of the class of multimedia and image processing applications that the *Spark* parallelizing high-level synthesis framework has been developed for. In this tutorial, we show how "C" code downloaded from the Internet can be synthesized using *Spark* and the resulting VHDL can be synthesized using commercial logic synthesis tools.

2.1 Downloading and Setting up the Tutorial

The tutorial is part of the *Spark* distribution that can be found at the *Spark* website [2] at:

http://mesl.ucsd.edu/spark

Choose the appropriate distribution based on the operating system you want to work on. The distribution will be named "spark-[OS]-[Version].tar.gz". So let us say you are interested the 1.1 version for the Linux platform, you will download the file "spark-linux-1.1.tar.gz". For the Windows distribution, we ship a ZIP archive named "spark-win32-[Version].zip".

After downloading this file, gunzip and untar the file as follows:

```
gunzip spark-linux-1.1.tar.gz
tar xvf spark-linux-1.1.tar.gz
OR
unzip spark-win32-1.1.zip
```

Uncompressing (gunzip and untar) the distribution will create the following directory structure:

```
spark-[OS]-[Version]/
                     bin/
                     include/
                     tutorial/
                     spark-setup.csh
                     spark-setup.sh
```

where [OS] is the operating system for which you downloaded the distribution and [Version] is the *Spark* version. To begin with source the "spark-setup.scr" script as follows:

```
source spark-setup.csh   # if you are using csh/tcsh shell
. spark-setup.sh         # if you are using sh/bash shell
```

For the Windows distribution, if you are using *Spark* under CYGWIN or MSYS/MINGW, then you can source the spark-setup.sh file, else for native Windows, you can run the batch file "spark-setup.bat".

The "spark-setup" script sets up the path to the *Spark* executable, along with some environment variables and some default include paths required by *Spark*. The include directory contains standard "C" include files such as stdio.h et cetera that are included by some input applications (including our tutorial). However, note that for the Windows distribution, these include files are not useful. If you do want to include system files such as "stdio.h" et cetera, please specify the path to these files using the environment variable SPARK_INCLUDES or using the command-line flag "-I /path/to/include".

The directory "tutorial" contains the *Spark* tutorial. The contents of this directory are:

```
tutorial/
        mpeg_play/
        synops/
        README.txt
        README-win32.txt
```

The directory *mpeg_play* contains the Berkeley MPEG player distribution. We will next demonstrate how to synthesize the motion estimation algorithm from this MPEG player. The *synops* directory contains the Synopsys logic synthesis scripts for synthesizing the output VHDL generated by *Spark*. We will come back to this in Section 2.5.

2.2 Synthesizing the Motion Compensation Algorithm

We choose the Motion Compensation algorithm from the MPEG-1 player for synthesis since this is one of most computationally expensive parts of the decoder. In the Berkeley *mpeg_play* distribution, this algorithm can be found in the file *motionvector.c*. An overview of synthesis of motion compensation algorithm by *Spark* is shown in Figure 2.1. In this figure, the motion compensation algorithm corresponds to the *motionvector.c* file.

Before we can synthesize the *motionvector.c* file using *Spark*, we have to check if the C code in this file is synthesizable. We find that the *motionvector.c* file has two functions *ComputeForwVector* and *ComputeBackVector*. However, one of the inputs of the *ComputeForwVector* and *ComputeBackVector* functions is a structure $VidStream * the_stream$. These functions then call the *ComputeVector* function (declared as a $\#define$) with the relevant members of the struct $VidStream * the_stream$.

Since *Spark* does not currently support structs (see User Manual), we split the *motionvector.c* file into two files: *motionvector_callsSpark.c* and *motionvector_forSpark.c*. As the names suggest, the first file calls the second file. We create two functions corresponding to the *ComputeForwVector* and *ComputeBackVector* functions in the

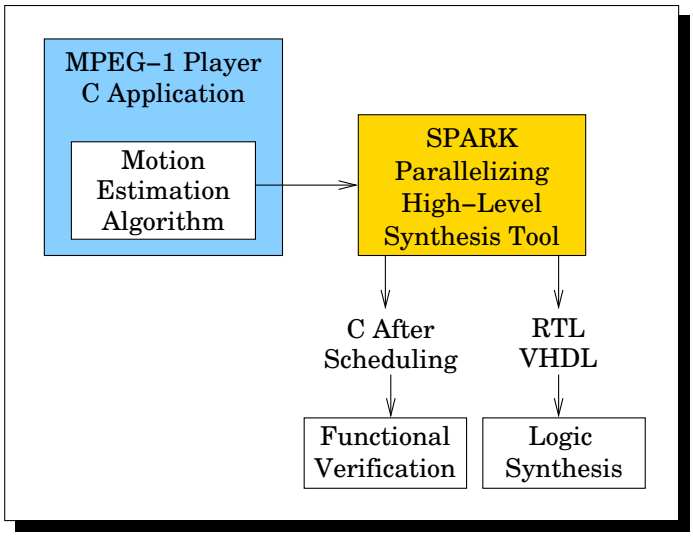

Figure 2.1. *Partitioning of the MPEG player application and synthesis of the motion compensation algorithm by the* Spark *high-level synthesis tool.* Spark *produces both RTL VHDL and behavioral C as output. The C can be used for functional verification against the input C, and the VHDL can be synthesized using commercial logic synthesis tools.*

motionvector_forSpark.c file that are synthesizable by *Spark*. We call these functions *ComputeForwVector_Modified* and *ComputeBackVector_Modified*. These functions take as input the members of the struct *VidStream*∗*the_stream* that are required by the *ComputeVector* function. The *motionvector_callsSpark.c* file now contains the original *ComputeForwVector* and *ComputeBackVector* functions that now call the modified functions with the appropriate inputs taken from the members of the *VidStream*∗*the_stream* struct.

We can now synthesize the *motionvector_forSpark.c* file by sourcing the script *runTut* given in the *mpeg_play* directory as follows:

```
source runTut.csh   # for csh/tcsh shell
. runTut.sh         # for sh/bash shell
runTut.bat          # native Windows batch file
```

This script first checks to see if it can find the *Spark* binary in the shell path (this check is not done in the Windows distribution; please make sure *Spark* is in your path). It then runs *Spark* on the *motionvector_forSpark.c* file by invoking the following command:

spark -m -hli -hcs -hcp -hdc -hs -hcc -hvf -hb -hec motionvector_forSpark.c

Note that: For the windows distribution, the *motionvector_forSpark.c* is modified as *motionvector_forSpark_win32.c* This is to remove "#include" of system files such as stdio.h. For the rest of this tutorial, whenever *motionvector_forSpark.c* is used, it refers to the file *motionvector_forSpark_win32.c* in the Windows distribution.

This command does loop-invariant code motion (-hli), common sub-expression elimination (-hcs), copy propagation (-hcp), dead code elimination (-hdc), scheduling (-hs), output C-code generation (-hcc), output RTL VHDL code generation (-hvf), resource binding (-hb), and counts the execution cycles (-hec).

Invoking *Spark* with these command-line flags produces three outputs: (a) graphical outputs corresponding to Hierarchical Task Graphs (HTGs), Control Flow Graphs (CFGs) and Data Flow Graphs(DFG), (b) "C" corresponding to the scheduled design, and (c) VHDL corresponding to the design after scheduling, resource binding and control-synthesis.

To schedule, bind, perform control synthesis, and generate output VHDL, *Spark* reads two other files besides the *motionvector_forSpark.c* file. These are the *default.spark* and the *Priority.rules* files in the *mpeg_play* directory. The *default.spark* file is a hardware description file that contains information on timing, range of the various data types, and the list of resources allocated to schedule the design (resource library). The *Priority.rules* file contains the synthesis scripts that control the transformations applied to the design description. Detailed information about these files is given in the *Spark* user manual.

2.3 Functional Verification of Output C Generated by *Spark*

Besides RTL VHDL, *Spark* also generates a behavioral C file that corresponds to the transformed and scheduled design. Clearly, this C file does not have any of the timing (clock boundary) and structural information, but it does have the code transformed by the optimizations applied by *Spark*.

Note that: This functional verification is **not** done by the Windows distribution of *Spark*. This is because the *mpeg_play* MPEG player uses X11 libraries and include files for executing. It is thus not portable to Windows (as far as we know). Users of the Windows distribution can skip the rest of this section.

The *runTut.csh/sh* does functional verification using the output C code against the input C code by calling three targets from the Makefile of *mpeg_play*. These calls to "make" produce three executables of the *mpeg_play* distribution for the following three motion estimation files:

- *motionvector.c*: Original motion compensation algorithm shipped with the Berkeley *mpeg_play* software. This produces the executable *mpeg_play-orig*.
- *motionvector_forSpark.c*: motion compensation algorithm modified to make it synthesizable by *Spark*. This produces the executable *mpeg_play-forspark*.
- *./output/motionvector_forSpark_sparkout.c*: The C output file for the motion compensation algorithm produced by *Spark* after scheduling/synthesis. This produces the executable *mpeg_play-afterspark*.

You can now test that the three executables work and produce the same results by playing the MPEG video file *smoker.mpg* provided with the tutorial. This can be done as follows:

```
mpeg_play-orig smoker.mpg
mpeg_play-forspark smoker.mpg
mpeg_play-afterspark smoker.mpg
```

In this way, we can do functional verification of the output C generated by *Spark* with the input C. This gives us some confidence that at least the transformations applied by *Spark* have not changed the functionality of the application.

2.4 Viewing the Graphical Output of *Spark*

Spark produces output graphs for each function in the "C" input file, before and after scheduling. The format that *Spark* uses for the output graphs is that of AT&Ts *Graphviz* tool [3]. See the *Spark* User Manual for more details on the output graphs generated.

You can view the graphs for the input and output (scheduled) *motionvector_forSpark.c* file by running the following commands:

```
dotty output/HTG_motionvector_forSpark_c_ComputeForwVector_Modified.dotty
dotty output/HTG_motionvector_forSpark_c_ComputeForwVector_Modified_sched.dotty
```

for the *ComputeForwVector* function.

```
dotty output/HTG_motionvector_forSpark_c_ComputeBackVector_Modified.dotty
dotty output/HTG_motionvector_forSpark_c_ComputeBackVector_Modified_sched.dotty
```

for the *ComputeBackVector* function.

2.5 Logic Synthesis of VHDL Output of *Spark*

Running *Spark* on the motion compensation algorithm by sourcing the *runTut.csh/sh* script also produces an output VHDL file. This VHDL file *./output/motionvector_forSpark_spark_rtl.vhd* can be synthesized by Synopsys Design Compiler.

Change directory to "tutorial/synops". The contents of the *synops* directory are as follows:

```
synops/
       dbs/
       reports/
       scripts/
       src/
```

The *runTut.csh/sh* script copies the VHDL file *./output/motionvector_forSpark_spark_rtl.vhd* to the *src* directory here. The *scripts* directory contains the Synopsys synthesis script *motionvector_forSpark_spark_rtl.scr*. This script can be used for Design Compiler as follows:

```
dc_shell -f motionvector_forSpark_spark_rtl.scr > dc_out.log
```

This script synthesizes the VHDL file and stores the generated reports in the *synops/reports* directory and the generated database files in the *synops/dbs* directory.

We have set the execution time of the multiplier as 2 cycles in the hardware description file distributed with the *Spark* tutorial. You can alter this cycle time in the *default.spark* file that is in the *mpeg_play* directory under the section titled [*Resources*]. More information on the format of the *default.spark* file is given in the user manual.

2.6 Logic Synthesis using non-Synopsys Logic Synthesis Tools

For logic synthesis using tools from other vendors, you can instruct *Spark* to generate VHDL that is not vendor specific as follows:

- Set the "PrintSynopsysVHDL" to *false* in the "[OutputVHDLRules]" section of the default.spark. This removes references to Synopsys libraries and DesignWare functions from the VHDL code. For example, "library Synopsys, DWARE;" is removed.
- You will have to explicitly instantiate multi-cycle components such as the multiplier and divider (if any) from the standard cell library of your technology vendor. This has to be done in the architecture description of *res_MUL* and *res_DIV* in the VHDL code. Note that, in this tutorial we use a two cycle multiplier.

Also, you will have to write synthesis scripts for your logic synthesis vendor.

Bibliography

[1] The berkeley mpeg player. http://bmrc.berkeley.edu/frame/research/mpeg/mpeg_play.html.

[2] SPARK parallelizing high-level synthesis framework website. http://www.cecs.uci.edu/~spark.

[3] AT&T Research Labs. Graphviz - Open source graph drawing software. http://www.research.att.com/sw/tools/graphviz/.